



Cursores Implícitos

Os exemplos utilizados ao longo do texto, salvo menção em contrário, foram adaptados de GROFF, J. R., WEINBERG, P. N. e OPPEL, A. J. **SQL: The Complete Reference**, relacionado nas referências bibliográficas. Os *scripts* para criação e população das tabelas encontram-se no material de apoio.

CURSORES IMPLÍCITOS

De forma bastante simplista e resumida, cursores (*cursors*) são estruturas onde são armazenados os resultados e informações relativos à execução de comandos DML. Há dois tipos de cursores: implícitos e explícitos. Cursores implícitos são automaticamente criados e gerenciados pelo SGBDR quando são executados os comandos INSERT, UPDATE, DELETE, SELECT INTO ou SELECT BULK COLLECT INTO. Embora o SGDBR Oracle crie cursores implícitos para todos comandos DML, sua utilização mais comum é a partir das duas últimas formas do comando SELECT. A seguir, alguns exemplos de cursores implícitos.





```
DECLARE
  linha filiais%ROWTYPE;
  regiao filiais.regiao%TYPE;
  cidade filiais.cidade%TYPE;
  filial_id filiais.filial_id%TYPE;
  vendas filiais.vendas%TYPE;
BEGIN
  SELECT F.regiao, F.cidade, F.filial_id, F.vendas INTO regiao,
cidade, filial_id, vendas
    FROM filiais F ORDER BY F.regiao, F.vendas DESC FETCH
NEXT 1 ROW ONLY;
  DBMS_OUTPUT.PUT_LINE('A filial com o maior volume de vendas
da regiao ' || regiao || ' é ' ||
    cidade || ' (id = ' || filial_id || '), com ' || TO_CHAR(vendas,
'B9G999G999D00'));
  /* A estrutura da cláusula INTO deve ter o mesmo número de campos
e os mesmos tipos da tabela   */
  SELECT * INTO linha FROM filiais F ORDER BY F.regiao, F.vendas
DESC FETCH NEXT 1 ROW ONLY;
  DBMS_OUTPUT.PUT_LINE('A filial com o maior volume de vendas
da regiao ' || linha.regiao || ' é ' ||
    linha.cidade || ' (id = ' || linha.filial_id || '), com ' ||
    TO_CHAR(linha.vendas, 'B9G999G999D00'));
END;
```

Após a execução, o resultado exibido é:

```
A filial com o maior volume de vendas da regiao Leste é Chicago (id =
12), com    735.042,00
A filial com o maior volume de vendas da regiao Leste é Chicago (id =
12), com    735.042,00
```

No exemplo anterior, foram apresentadas duas formas distintas, porém equivalentes, de utilização dos resultados de uma consulta em um programa PL/SQL. Em ambas, o resultado consistia em uma única linha. Um erro ocorreria se a consulta retornasse mais linhas.

Na primeira forma, os valores de quatro colunas foram atribuídos a quatro variáveis. Observe que tanto o número quanto o tipo das

variáveis devem ser iguais (ou equivalentes, no caso do tipo) aos das colunas selecionadas. Na segunda, uma linha inteira foi atribuída a uma variável tipo RECORD, com estrutura idêntica à da tabela.

Os dois erros mais comuns que podem ocorrer com cursores implícitos são *ORA-01403 data not found* (a consulta não retornou qualquer linha) e *ORA-01422 exact fetch returns more than requested number of rows* (a consulta retornou mais de uma linha). É aconselhável que ambos os casos sejam tratados dentro do próprio procedimento. No exemplo a seguir, a *procedure* retorna o nome da filial cujo *rep_id* do gerente é passado como parâmetro.

```
CREATE OR REPLACE PROCEDURE dados_filial(
   rep_id IN filiais.rep_id%TYPE,
   filial OUT VARCHAR2,
   status OUT NUMBER
)
AS
   linha filiais%ROWTYPE;
BEGIN
   status := 0;
   filial := '';
   SELECT * INTO linha FROM filiais F WHERE F.rep_id = rep_id;
   filial := linha.filial_id || ' - ' || linha.cidade || ', ' || linha.regiao;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      status := -1;
   WHEN TOO_MANY_ROWS THEN
      status := -2;
   WHEN OTHERS THEN
      status := -3;
END;

DECLARE
   filial VARCHAR2(200);
   status NUMBER;
BEGIN
  FOR rep_id IN 100..110 LOOP
     dados_filial(rep_id, filial, status);
     DBMS_OUTPUT.PUT_LINE('(' || status || ') ' || rep_id || ' | ' || CASE
status WHEN 0 THEN filial
       WHEN -1 THEN 'Não gerencia qualquer filial' WHEN -2 THEN
'Gerencia duas ou mais filiais'
       ELSE 'Erro desconhecido' END);
   END LOOP;
END;
```

Após a execução, o resultado exibido é:

(-1) 100 | Não gerencia qualquer filial
(-1) 101 | Não gerencia qualquer filial
(-1) 102 | Não gerencia qualquer filial
(-1) 103 | Não gerencia qualquer filial
(0) 104 | 12 - Chicago, Leste
(0) 105 | 13 - Atlanta, Leste
(0) 106 | 11 - New York, Leste
(-1) 107 | Não gerencia qualquer filial
(-2) 108 | Gerencia duas ou mais filiais
(-1) 109 | Não gerencia qualquer filial
(-1) 110 | Não gerencia qualquer filial



EXECUÇÃO EM MASSA (BULK)

PL/SQL permite tratar resultados de consultas com múltiplas linhas. As cláusulas BULK COLLECT INTO (SELECT) e FORALL (INSERT, UPDATE e DELETE) permitem tratar estes casos. Uma versão da *procedure DADOS_FILIAL*, alterada para tratar os casos em que um representante gerencia mais de uma filial, é mostrada a seguir.

```
1  CREATE OR REPLACE PACKAGE meus_tipos
2  IS
3    TYPE tipo_dados_filiais IS TABLE OF filiais%ROWTYPE;
4  END;
5
6  CREATE OR REPLACE PROCEDURE filiais_por_gerente(
7    rep_id IN filiais.rep_id%TYPE,
8    dados_filiais OUT meus_tipos.tipo_dados_filiais,
9    status OUT NUMBER
10 )
11 AS
12 BEGIN
13   status := 0;
14   SELECT * BULK COLLECT INTO dados_filiais FROM filiais F WHERE F.rep_id =
15     filiais_por_gerente.rep_id;
16 EXCEPTION
17   WHEN OTHERS THEN
18     status := SQLCODE;
19 END;
20
21 DECLARE
22   rep_id filiais.rep_id%TYPE;
23   dados_filiais meus_tipos.tipo_dados_filiais;
24   status NUMBER;
25 BEGIN
26   FOR rep_id IN 100..110 LOOP
27      filiais_por_gerente(rep_id, dados_filiais, status);
28      IF status = 0 THEN
29          DBMS_OUTPUT.PUT_LINE('rep_id: ' || rep_id || ' | dados_filiais.COUNT: ' ||
30            dados_filiais.COUNT || '  ');
31          FOR i IN 1..dados_filiais.COUNT LOOP
32              DBMS_OUTPUT.PUT_LINE('(' || status || ') ' || rep_id || ' | ' ||
33                 dados_filiais(i).filial_id || ' - ' || dados_filiais(i).cidade ||
34                 ', ' || dados_filiais(i).regiao);
35          END LOOP;
36       ELSE
37          DBMS_OUTPUT.PUT_LINE('(' || status || ') Erro não identificado');
38       END IF;
39    END LOOP
40 END;
```

Das linhas 1 a 4 é criado o tipo *tipo_dadosfiliais*, uma tabela aninhada de registros, que receberá o resultado da consulta. Este tipo é global ao esquema. O parâmetro de saída *dados_filiais* (linha 8), agora pode receber 0 ou mais linhas de resultado. Observe que o tipo é qualificado. O comando SELECT BULK COLLECT INTO, nas linhas 14 e 15, atribui o resultado da consulta na tabela aninhada *dados_filiais.* Por se tratar de um cursor implícito, não são necessárias a inicialização ou inserção de elementos em *dados_filiais*. Isto é feito automaticamente após a execução da consulta. Observe também que não é mais necessário incluir tratamento para os erros ORA-01403 e ORA-01422. Para saber a quantidade de linhas retornadas

pela consulta, utiliza-se o atributo COUNT. Após a execução, o resultado exibido é:



```
rep_id: 100 | dados_filiais.COUNT: 0
rep_id: 101 | dados_filiais.COUNT: 0
rep_id: 102 | dados_filiais.COUNT: 0
rep_id: 103 | dados_filiais.COUNT: 0
rep_id: 104 | dados_filiais.COUNT: 1
(0) 104 | 12 - Chicago, Leste
rep_id: 105 | dados_filiais.COUNT: 1
(0) 105 | 13 - Atlanta, Leste
rep_id: 106 | dados_filiais.COUNT: 1
(0) 106 | 11 - New York, Leste
rep_id: 107 | dados_filiais.COUNT: 0
rep_id: 108 | dados_filiais.COUNT: 2
(0) 108 | 22 - Denver, Oeste
(0) 108 | 21 - Los Angeles, Oeste
rep_id: 109 | dados_filiais.COUNT: 0
rep_id: 110 | dados_filiais.COUNT: 0
```

É possível limitar o número de linhas atribuídas pelo comando SELECT BULK COLLECT INTO colocando-se a cláusula LIMIT *n* ao final do comando, onde *n* é o número máximo desejado.

O comando FORALL permite que múltiplas linhas sejam inseridas, excluídas ou alteradas de uma só vez. Sua forma geral é:

```
FORALL índice IN
[limite_inferior .. limite_superior]
[SAVE EXCEPTIONS]
comando_sql;
```

A melhor forma de entender o funcionamento do comando FORALL é através de exemplos. Deseja-se criar duas novas *procedures* para manutenção da tabela *PRODUTOS*. A primeira, *insere_produtos*, recebe como parâmetro uma lista com novos produtos a serem inseridos e outra, *atualiza_precos*, recebe *como* parâmetro o código do fornecedor e uma lista com códigos de produtos e percentuais de reajuste no preço.

```
CREATE OR REPLACE PACKAGE meus_tipos
IS
   TYPE tipo_dados_filiais IS TABLE OF filiais%ROWTYPE;
   TYPE tipo_tab_produtos IS TABLE OF produtos%ROWTYPE;
   TYPE tipo_tab_produto_id IS TABLE OF
produtos.produto_id%TYPE;
   TYPE tipo_tab_fornec_id IS TABLE OF produtos.fornec_id%TYPE;
   TYPE tipo_tab_number IS TABLE OF NUMBER;
END;

CREATE OR REPLACE PROCEDURE insere_produtos (
   registros meus_tipos.tipo_tab_produtos
)
AS
BEGIN
   /* Método 1: inserção de um registro de cada vez */
   FOR i IN registros.FIRST .. registros.LAST LOOP
      INSERT INTO produtos VALUES registros(i);
   END LOOP;
END;

CREATE OR REPLACE PROCEDURE atualiza_precos (
   fornec_id   meus_tipos.tipo_tab_fornec_id,
   produto_id  meus_tipos.tipo_tab_produto_id,
   perc_ajuste meus_tipos.tipo_tab_number
)
AS
BEGIN
   /* Método 1: atualização de um registro de cada vez */
   FOR i IN produto_id.FIRST .. produto_id.LAST LOOP
      UPDATE produtos SET preco = preco * (1 + perc_ajuste(i))
      WHERE fornec_id = atualiza_precos.fornec_id(i) AND
       produto_id = atualiza_precos.produto_id(i);
   END LOOP;
END;
```

Os blocos de comandos que preparam os parâmetros e chamam as duas *procedures* encontram-se a seguir.



```
DECLARE
   produtos meus_tipos.tipo_tab_produtos;
BEGIN
   produtos := meus_tipos.tipo_tab_produtos();
   produtos.EXTEND(3);
   produtos(1).fornec_id := 'IMM'; produtos(1).produto_id := '771C';
   produtos(1).descricao := '100-lb brace'; produtos(1).preco := 390;
   produtos(1).qtd_disponivel := 50;
   produtos(2).fornec_id := 'ACI'; produtos(2).produto_id := '4100W';
   produtos(2).descricao := 'Widget Tester'; produtos(2).preco := 450;
   produtos(2).qtd_disponivel := 50;
   produtos(3).fornec_id := 'REI'; produtos(3).produto_id := '2A44C';
   produtos(3).descricao := 'Central Hinge'; produtos(3).preco := 4750;
   produtos(3).qtd_disponivel := 20;
   insere_produtos(produtos);
END;

DECLARE
   fornec_id   meus_tipos.tipo_tab_fornec_id;
   produto_id  meus_tipos.tipo_tab_produto_id;
   perc_ajuste meus_tipos.tipo_tab_number;
BEGIN
   fornec_id   := meus_tipos.tipo_tab_fornec_id();
   produto_id  := meus_tipos.tipo_tab_produto_id();
   perc_ajuste := meus_tipos.tipo_tab_number();
   fornec_id.EXTEND(3);
   produto_id.EXTEND(3);
   perc_ajuste.EXTEND(3);
   fornec_id(1):= 'IMM'; produto_id(1) := '771C';  perc_ajuste(1) := 0.05;
   fornec_id(2):= 'ACI'; produto_id(2) := '4100W'; perc_ajuste(2) := 0.025;
   fornec_id(3):= 'REI'; produto_id(3) := '2A44C'; perc_ajuste(3) := -0.05;
   atualiza_precos(fornec_id, produto_id, perc_ajuste);
END;
```

Nesta primeira versão, ambas as *procedures* executam os comandos DML dentro de um *loop*, um comando por vez. Isto não é muito eficiente, já que o SGBDR deve alternar entre o módulo de execução SQL e PL/SQL a cada iteração do *loop*.

No exemplo a seguir, as duas *procedures* são modificadas para utilizarem o comando FORALL. Embora tenha a mesma forma um *loop* tradicional, o FORALL não é um comando de repetição. Na realidade, ele não é um comando PL/SQL, mas sim SQL (é executado pelo módulo SQL do SGBDR). Este comando "monta" os comandos SQL e os envia de uma só vez ao módulo SQL, evitando assim muitas trocas de contexto.

```
CREATE OR REPLACE PROCEDURE insere_produtos2 (
   registros meus_tipos.tipo_tab_produtos
)
AS
BEGIN
   /* Método 2: utilizando o comando FORALL */
   FORALL i IN registros.FIRST .. registros.LAST
      INSERT INTO produtos VALUES registros(i);
END;

CREATE OR REPLACE PROCEDURE atualiza_precos2 (
   fornec_id   meus_tipos.tipo_tab_fornec_id,
   produto_id  meus_tipos.tipo_tab_produto_id,
   perc_ajuste meus_tipos.tipo_tab_number
)
AS
BEGIN
   /* Método 2: utilizando o comando FORALL */
   FORALL i IN produto_id.FIRST .. produto_id.LAST
      UPDATE produtos SET preco = preco * (1 + perc_ajuste(i))
      WHERE fornec_id = atualiza_precos2.fornec_id(i) AND
produto_id =
        atualiza_precos2.produto_id(i);
END;
```

ATRIBUTOS DE CURSORES IMPLÍCITOS

O SGBDR Oracle permite que algumas informações a respeito do último cursor implícito executado sejam acessadas. Estas

informações são armazenadas por atributos pré-definidos que pertencem à estrutura SQL, gerada automaticamente para cada comando DML executado. São eles:

Nome	Descrição
SQL%FOUND	Retorna TRUE se pelo menos uma linha foi alterada (INSERT, UPDATE e DELETE) ou se uma consulta retornou uma (SELECT INTO) ou mais (SELECT BULK COLLECT INTO) linhas.
SQL%NOTFOUND	Retorna TRUE se nenhuma linha foi alterada (INSERT, UPDATE e DELETE) ou se uma consulta não retornou qualquer linha (= NOT SQL%FOUND).
SQL%ROWCOUNT	Retorna o número de linhas alteradas (INSERT, UPDATE e DELETE) ou o número de linhas retornadas em uma consulta com sucesso (SELECT INTO).
SQL%ISOPEN	Sempre retorna FALSE para cursores implícitos.

```
CREATE OR REPLACE FUNCTION bool_to_text(x IN BOOLEAN)
RETURN VARCHAR2
IS
BEGIN
   IF x THEN
      RETURN 'TRUE';
   ELSIF NOT x THEN
      RETURN 'FALSE';
   ELSE
      RETURN 'NULL';
   END IF;
END;

DECLARE
   TYPE tipo_pedidos IS TABLE OF pedidos%ROWTYPE;
   p1 tipo_pedidos;
   p2 pedidos%ROWTYPE;
BEGIN
   SELECT * INTO p2 FROM pedidos WHERE pedido_id = 113024;
   DBMS_OUTPUT.PUT_LINE(bool_to_text(SQL%FOUND) || '    ' ||
bool_to_text(SQL%NOTFOUND) || '  ' ||
     SQL%ROWCOUNT);
   SELECT * BULK COLLECT INTO p1 FROM pedidos WHERE
pedido_id BETWEEN 113000 AND 114000;
   DBMS_OUTPUT.PUT_LINE(bool_to_text(SQL%FOUND) || '    ' ||
bool_to_text(SQL%NOTFOUND) || '  ' ||
     SQL%ROWCOUNT);
   SELECT * BULK COLLECT INTO p1 FROM pedidos WHERE
pedido_id > 114000;
   DBMS_OUTPUT.PUT_LINE(bool_to_text(SQL%FOUND) || '    ' ||
bool_to_text(SQL%NOTFOUND) || '  ' ||
     SQL%ROWCOUNT);
   SELECT * BULK COLLECT INTO p1 FROM pedidos WHERE
pedido_id > 113060;
   FOR i IN p1.FIRST .. P1.LAST LOOP
      p1(i).qtd := p1(i).qtd * 2;
   END LOOP;
   FORALL i IN p1.FIRST .. p1.LAST
```

Após a execução do bloco de comandos, o resultado apresentado é:



```
TRUE   FALSE  1
TRUE   FALSE  18
FALSE  TRUE   0
TRUE   FALSE  3
TRUE   FALSE  3
```

TRATAMENTO DE EXCEÇÕES

Erros podem ocorrer durante a execução em massa de comandos DML. A cláusula SAVE EXCEPTIONS evita que o processamento seja interrompido, salvando informações dos comandos que causaram erro e fazendo com que uma única exceção seja gerada ao final do processamento. Estas informações são salvas na coleção SQL%BULK_EXCEPTIONS, cujos elementos são RECORDS com dois campos: ERROR_INDEX e ERROR_CODE. ERROR_INDEX informa o índice do elemento da coleção que causou o erro e ERROR_CODE informa o código do erro. O exemplo a seguir, que utiliza as mesmas *procedures* dos exemplos anteriores, ilustra seu funcionamento.

```
CREATE OR REPLACE PROCEDURE insere_produtos3 (
   registros meus_tipos.tipo_tab_produtos
)
AS
BEGIN
   /* Método 3: utilizando o comando FORALL com tratamento de
exceções */
   FORALL i IN registros.FIRST .. registros.LAST
      SAVE EXCEPTIONS
      INSERT INTO produtos VALUES registros(i);
EXCEPTION
   WHEN  OTHERS THEN
      print_excecoes;
END;

CREATE OR REPLACE PROCEDURE atualiza_precos3 (
   fornec_id   meus_tipos.tipo_tab_fornec_id,
   produto_id  meus_tipos.tipo_tab_produto_id,
   perc_ajuste meus_tipos.tipo_tab_number
)
AS
BEGIN
   /* Método 3: utilizando o comando FORALL com tratamento de
exceções */
   FORALL i IN produto_id.FIRST .. produto_id.LAST
      SAVE EXCEPTIONS
      UPDATE produtos SET preco = preco * (1 + perc_ajuste(i))
      WHERE fornec_id = atualiza_precos3.fornec_id(i) AND
produto_id =
         atualiza_precos3.produto_id(i);
EXCEPTION
   WHEN  OTHERS THEN
      print_excecoes;
END;

CREATE OR REPLACE PROCEDURE print_excecoes
AS
```

No bloco anônimo, o primeiro e segundo registros são iguais, o que ocasionará violação de unicidade em *PRODUTOS*. A seção de

tratamento de exceções chama a *procedure print_excecao* que manipula a coleção SQL%BULK_EXCEPTIONS. Observe que não é necessário passar SQL%BULK_EXCEPTIONS como parâmetro para *print_excecao*, já que esta coleção é global. O resultado, após a execução do bloco anônimo, é mostrado a seguir.

```
Total de erros: 1
Erro 1: iteração n. 2, erro n. ORA-00001: unique constraint (.) violated
```

CURSORES EXPLÍCITOS

Cursores explícitos são declarados como tal em blocos de comando PL/SQL. Diferentemente dos cursores implícitos, é responsabilidade do usuário abrir, executar a consulta SQL e fechar o cursor. Cursores explícitos são declarados utilizando-se a diretiva CURSOR na seção de declaração do bloco. Os comandos OPEN, FETCH e CLOSE são utilizados, respectivamente para abrir, executar e fechar o cursor. As formas gerais da diretiva CURSOR e dos comandos de manipulação de cursores são mostradas a seguir.

```
CURSOR nome_cursor [(parâmetro1 [, parâmetro2] ...)]
[RETURN especificação_retorno]
IS SELECT_cláusulas_select
[FOR UPDATE [OF lista_colunas]];

OPEN nome_cursor [(parâmetro1 [, parâmetro2] ...)];

FETCH nome_cursor INTO variável_retorno;

CLOSE nome_cursor;
```

- Os parâmetros *parâmetro1*, *parâmetro2*, ..., se presentes, podem ser utilizados no comando SELECT;

- A cláusula RETURN especifica o tipo retornado por cada FETCH executado. *Especificação_retorno* pode ser um tipo RECORD definido ou um tipo %ROWTYPE. Se não for especificado, o retorno é do tipo RECORD com os campos iguais às colunas referenciadas no comando SELECT;

- Cursores explícitos são utilizados apenas com o comando SELECT. *Cláusulas_select* especifica o comando SELECT associado ao cursor;

- A cláusula FOR UPDATE é utilizada quando se deseja garantir que as linhas que compõem o resultado da consulta não sejam modificadas por outro usuário;

- O comando OPEN abre o cursor e executa o comando SELECT correspondente. Os parâmetros devem coincidir em número e tipo com o que foi especificado na declaração do cursor. Ao abrir o cursor, é também criado um ponteiro para a linha a ser trazida pelo primeiro FETCH. A tentativa de se abrir um cursor já aberto causará erro;

- O comando FETCH retorna os valores da linha apontada pelo ponteiro de linha e avança o ponteiro para a próxima linha, se houver. O resultado da consulta e *variável_retorno* devem ser do mesmo tipo. A tentativa de se executar um FETCH um cursor fechado causará erro;

- O comando CLOSE fecha o cursor e libera a memória e as linhas da tabela (caso a opção FOR UPDATE tenha sido usada) usadas por ele. A tentativa de se fechar um cursor já fechado causará erro;

Os atributos de cursores explícitos são armazenados na estrutura SQL, gerada automaticamente após a execução dos comandos OPEN, FETCH e CLOSE. São eles:

Nome	Descrição
cursor%FOUND	Retorna TRUE se o comando FETCH retornou uma nova linha.
cursor%NOTFOUND	Retorna TRUE se o comando FETCH não retornou linha alguma (= NOT SQL%FOUND).
cursor%ROWCOUNT	Retorna o número de linhas retornadas pelo comando FETCH até aquele momento.
cursor%ISOPEN	Retorna TRUE se o cursor estiver aberto.

No exemplo a seguir, a *procedure relatorio_pedidos* recebe como parâmetro o código do cliente (*cliente_id*) e imprime o relatório de pedidos deste cliente.

```
create or replace PROCEDURE relatorio_pedidos (
    cliente_id pedidos.cliente_id%TYPE
)
AS
    CURSOR cur_pedidos (pedido_id pedidos.cliente_id%TYPE) IS
        SELECT Pe.pedido_id AS PEDIDO, Pe.data AS DATA, C.empresa as CLIENTE,
            Pr.descricao AS PRODUTO, Pe.qtd AS QTD, Pe.VALOR as VALOR, Pe.qtd * Pe.valor AS TOTAL
        FROM pedidos Pe, clientes C, produtos Pr
        WHERE Pe.cliente_id = cur_pedidos.pedido_id AND Pe.cliente_id = C.cliente_id AND
            Pe.produto_id = Pr.Produto_id;
    reg_pedidos cur_pedidos%ROWTYPE;
    tipo_colunas meus_tipos.tipo_tab_varchar;
BEGIN
    tipo_colunas := meus_tipos.tipo_tab_varchar('N', 'D', 'T', 'T', 'N', 'V', 'V');
    OPEN cur_pedidos(relatorio_pedidos.cliente_id);
    FETCH cur_pedidos INTO reg_pedidos;
    DBMS_OUTPUT.PUT_LINE('PEDIDO  DATA     CLIENTE              PRODUTO                   QTD' ||
        '         VALOR         TOTAL');
    DBMS_OUTPUT.PUT_LINE('======  =======  ==================== ==================== =======' ||
        ' ============= =============');
    WHILE cur_pedidos%FOUND LOOP
        DBMS_OUTPUT.PUT(TO_CHAR(reg_pedidos.PEDIDO, '999999') || ' ');
        DBMS_OUTPUT.PUT(TO_CHAR(reg_pedidos.DATA, 'DD/MM/YY') || ' ');
        DBMS_OUTPUT.PUT(reg_pedidos.CLIENTE || SUBSTR(RPAD(' ', 20, ' '), 1, 21 -
            LENGTH(reg_pedidos.CLIENTE)));
        DBMS_OUTPUT.PUT(reg_pedidos.PRODUTO || SUBSTR(RPAD(' ', 21, ' '), 1, 21 -
            LENGTH(reg_pedidos.PRODUTO)));
        DBMS_OUTPUT.PUT(TO_CHAR(reg_pedidos.QTD, '999999') || ' ');
        DBMS_OUTPUT.PUT(TO_CHAR(reg_pedidos.VALOR, '9G999G999D99') || ' ');
        DBMS_OUTPUT.PUT(TO_CHAR(reg_pedidos.TOTAL, '9G999G999D99') || ' ');
        DBMS_OUTPUT.NEW_LINE;
        FETCH cur_pedidos INTO reg_pedidos;
    END LOOP;
    CLOSE cur_pedidos;
END;
```

A diretiva CURSOR (linhas 5 a 10) é utilizada para se definir a estrutura do cursor *cur_pedidos*. Ele recebe um parâmetro de entrada, que é utilizado na cláusula WHERE do comando SELECT. Observe que a lista de colunas do SELECT pode ter codinomes e colunas calculadas. Na realidade, colunas calculadas devem ter necessariamente codinomes, caso contrário não será possível referenciá-las. Depois de definida a sua estrutura, é necessário criar uma variável para receber as linhas de *cur_pedidos*, o que é feito na linha 11. Recomenda-se sempre utilizar o tipo ancorado %ROWTYPE. A variável *tipo_colunas* informa, para fins de formatação apenas, o tipo de cada coluna que comporá o relatório. O cursor é aberto na linha 15, executando a consulta associada, a primeira linha do resultado é carregada em *reg_pedidos* na linha 16.

O *loop* entre as linhas 22 e 33 processa o relatório. Os campos do resultado da consulta são referenciados como os campos de um RECORD (por esta razão é necessário atribuírem-se codinomes às colunas calculadas). Observe que a condição de saída do *loop* é o atributo FOUND do cursor *cur_pedidos*. Os atributos FOUND e NOT_FOUND recebem, respectivamente os valores FALSE e TRUE quando é executado um comando FETCH e o ponteiro de linha aponta para além da última linha.

Finalmente, na linha 34, o cursor é fechado. Embora o SGBDR Oracle feche automaticamente, ao final de um bloco de comandos, os cursores nele definidos, é uma boa prática fechá-los explicitamente após o uso.

Após a execução do bloco anônimo, o resultado apresentado é (apenas o início e final do relatório são apresentados; o relatório completo encontra-se no material de apoio).

```
PEDIDO  DATA    CLIENTE           PRODUTO            QTD
VALOR       TOTAL
======  =======  ====================
==================== ======= ============
=============
 113012 11/01/08 JCP Inc.          Size 3 Widget          35
3.745,00    131.075,00
 113012 11/01/08 JCP Inc.          Handle                35     3.745,00
131.075,00
 113057 18/02/08 JCP Inc.          Widget Adjuster         24
600,00     14.400,00
 112975 12/10/07 JCP Inc.          Hinge Pin               6     2.100,00
12.600,00

PEDIDO  DATA    CLIENTE           PRODUTO            QTD
VALOR       TOTAL
======  =======  ====================
==================== ======= ============
=============
 112993 04/01/07 Fred Lewis Corp.     Ratchet Link           24
1.896,00     45.504,00

 ...    ...     ...                ...                ...     ...        ...

PEDIDO  DATA    CLIENTE           PRODUTO            QTD
VALOR       TOTAL
======  =======  ====================
==================== ======= ============
=============
 113051 10/02/08 Midwest Systems     Reducer                4
1.420,00      5.680,00
 113049 10/02/08 Midwest Systems     Reducer                2
776,00      1.552,00
 113013 14/01/08 Midwest Systems     Size 3 Widget            1
652,00       652,00
 113013 14/01/08 Midwest Systems     Handle                 1
652,00       652,00
 112992 04/11/07 Midwest Systems     Size 2 Widget           10
760,00      7.600,00

PEDIDO  DATA    CLIENTE           PRODUTO            QTD
```

A CLÁUSULA FOR UPDATE

Quando consultas são enviadas ao SGBDR (comandos SELECT), nenhuma tranca (*lock*) é colocada nas linhas resultantes. Em algumas situações é necessário trancar as linhas consultadas, como por exemplo quando se deseja alterar uma ou mais linhas do resultado em função dos valores retornados. A linguagem PL/SQL permite que isto seja feito através da cláusula FOR UPDATE do comando SELECT. Quando a cláusula FOR UPDATE está presente, as linhas do resultado da consulta são automaticamente trancadas para alteração. Outros usuários podem, no entanto, ler estas linhas em suas consultas.

A cláusula FOR UPDATE OF *lista_colunas*, utilizada em consultas envolvendo múltiplas tabelas, especifica que tabelas terão suas linhas trancadas. Qualquer tabela que tenha uma ou mais colunas em *lista_colunas* terá suas linhas trancadas.

O trancamento de linhas das tabelas permanece até que os comandos COMMIT ou ROLLBACK sejam executados, mesmo que o cursor ainda permaneça aberto. Após a execução de um destes dois comandos, não é possível mais executar o comando FETCH em cursores com a cláusula FOR UPDATE, já que outros usuários poderão alterar as linhas previamente trancadas, quebrando a integridade dos dados do cursor e das tabelas envolvidas. O erro ORA-01002: *fetch out of sequence* ocorre nestas situações. Obviamente, o comando CLOSE libera todas as linhas das tabelas utilizadas no respectivo cursor com FOR UPDATE.

No exemplo a seguir, a *procedure atualiza_pedidos* altera os pedidos em que a venda foi realizada por um representante diferente daquele atende o cliente, ou seja, *rep_id* de *CLIENTE* é diferente de *rep_id* de *PEDIDO* e *cliente_id* de ambas é o mesmo.

```
1  create or replace PROCEDURE atualiza_pedidos
2  AS
3      CURSOR cur_pedidos IS
4          SELECT P.pedido_id PEDIDO_ID, P.rep_id REP_ID_OLD, C.rep_id REP_ID_NEW
5          FROM pedidos P, clientes C, representantes R
6          WHERE P.cliente_id = C.cliente_id  AND P.rep_id = R.rep_id AND R.rep_id <> C.rep_id
7      FOR UPDATE OF P.rep_id;
8      linha_pedidos cur_pedidos%ROWTYPE;
9      rep_id_new representantes.rep_id%TYPE;
10 BEGIN
11     OPEN cur_pedidos;
12     LOOP
13         FETCH cur_pedidos INTO linha_pedidos;
14         EXIT WHEN cur_pedidos%NOTFOUND;
15         UPDATE pedidos SET rep_id = linha_pedidos.REP_ID_NEW
16             WHERE pedido_id = linha_pedidos.PEDIDO_ID;
17         SELECT rep_id INTO rep_id_new FROM pedidos WHERE pedido_id = linha_pedidos.PEDIDO_ID;
18         DBMS_OUTPUT.PUT_LINE('pedido_id: ' || linha_pedidos.PEDIDO_ID || ' rep_id_old: ' ||
19             linha_pedidos.REP_ID_OLD || ' rep_id_new: ' || rep_id_new);
20     END LOOP;
21     CLOSE cur_pedidos;
22 END;

   BEGIN
       atualiza_pedidos;
   END;
```

O cursor definido retorna as linhas de *PEDIDOS* onde a condição acima ocorre (linhas 3 a 7). É importante que a cláusula FOR UPDATE seja incluída, pois outro usuário poderia alterar as linhas lidas antes que *rep_id* fosse atualizado, provocando inconsistência nos dados. Observe que apenas a coluna *rep_id* de *PEDIDOS* foi incluída, pois esta é a única coluna que será alterada, provocando assim o trancamento das linhas correspondentes de *PEDIDOS* apenas. Se nenhuma coluna fosse especificada, todas as linhas afetadas de todas as tabelas envolvidas seriam trancadas.[1]

O *loop* entre as linhas 12 e 20 processa as alterações. Enquanto o cursor não é fechado, nenhum outro usuário consegue alterar as linhas de *PEDIDOS* trancadas. Após a execução do bloco anônimo, o resultado apresentado é:

```
pedido_id: 113055 rep_id_old: 101 rep_id_new: 109
pedido_id: 113012 rep_id_old: 101 rep_id_new: 103
pedido_id: 113042 rep_id_old: 101 rep_id_new: 104
pedido_id: 113024 rep_id_old: 108 rep_id_new: 102
```



A CLÁUSULA WHERE CURRENT OF

A cláusula WHERE CURRENT OF é utilizada juntamente com os comandos UPDATE e DELETE, em substituição à cláusula WHERE, indicando que a linha a ser alterada ou excluída é aquela que foi lida pelo último comando FETCH. Pode parecer que o uso desta cláusula não traz qualquer benefício, mas não é bem assim. Imagine um cursor com uma cláusula WHERE com dezenas de expressões (isto é relativamente comum em grandes sistemas). Para realizar qualquer alteração em uma linha lida, esta mesma condição teria que ser repetida nos comandos UPDATE ou DELETE. Qualquer alteração nas tabelas subjacentes, implicaria alterações em dois locais diferentes, aumentando o trabalho e a chance de erro.

No exemplo a seguir, a atualização de linhas da tabela *PEDIDOS* da *procedure atualiza_pedidos* é modificada utilizando a cláusula WHERE CURRENT OF (só é mostrada a parte alterada).

```
create or replace PROCEDURE atualiza_pedidos2
...
BEGIN
   OPEN cur_pedidos;
   LOOP
  ...
      UPDATE pedidos SET rep_id = linha_pedidos.REP_ID_NEW
      WHERE CURRENT OF cur_pedidos;
      ...
   END LOOP;
   CLOSE cur_pedidos;
END;
```

O resultado exibido é o mesmo que o do exemplo anterior, como era de se esperar.

USO DE CURSORES EM *PACKAGES*

O uso de cursores em *packages* pode representar uma poderosa ferramenta. Há duas formas de se declarar cursores em *packages*. Na primeira, utiliza-se a cláusula RETURN, deixando a declaração da consulta de criação para o corpo da *package*. Na segunda, a consulta de criação e colocada na própria especificação da *package*. Do ponto de vista das funcionalidades, ambas as formas são equivalentes. A única diferença é que na segunda, a consulta de criação não é exposta.

Antes de serem usados, cursores precisam ser abertos. Como os objetos de uma *package* são globais à sessão, deve-se sempre verificar se o cursor está aberto, antes de qualquer tentativa de abri-lo. O mesmo vale para o fechamento de cursores. Uma sugestão para endereçar o problema é incluir na *package procedures* que abrem e fecham o cursor. A seguir, são incluídas na *package pkg_filmes* duas *procedures, open_cur_filmes* (esta com *oveloading*) e *close_cur_filmes,* que cuidarão de abrir e fechar o cursor *cur_filmes* (apenas as partes alteradas são mostradas).

```
CREATE OR REPLACE PACKAGE pkg_filmes AS
   -- Declaração de tipos, cursores, exceções e variáveis expo[illegible] ao
mundo externo
   TYPE tipo_reg_filmes IS RECORD (
    ...
   CURSOR cur_filmes_diretor (diretor filmes.diretor%TYPE) IS
      SELECT * FROM FILMES WHERE diretor =
cur_filmes_diretor.diretor;
   CURSOR cur_filmes_pais (pais filmes.pais%TYPE) IS
      SELECT * FROM FILMES WHERE pais = cur_filmes_pais.pais;
   CURSOR cur_filmes_ano (ano_ini filmes.ano%TYPE, ano_fim
filmes.ano%TYPE) IS
      SELECT * FROM FILMES WHERE ano BETWEEN
cur_filmes_ano.ano_ini AND cur_filmes_ano.ano_fim;
   FILME_REPETIDO EXCEPTION;
    ...
   consultas  NUMBER := 0;
   cur_diretor CONSTANT NUMBER := 0;
   cur_pais CONSTANT NUMBER := 0;
   cur_ano CONSTANT NUMBER := 0;
   -- Overload de funções: duas funções com nomes iguais, porém
com parâmetros formais diferentes
   FUNCTION consulta_filme (filme_id filmes.filme_id%TYPE)
      RETURN tipo_reg_filmes;
    ...
   FUNCTION altera_filme (registro tipo_reg_filmes)
      RETURN NUMBER;
   PROCEDURE open_cur_filmes (que_cursor IN NUMBER,
fecha_e_reabre IN BOOLEAN,
      diretor_ou_pais IN VARCHAR2);
   PROCEDURE open_cur_filmes (que_cursor IN NUMBER,
fecha_e_reabre IN BOOLEAN,
      ano_ini IN NUMBER, ano_fim IN NUMBER);
   PROCEDURE fecha_cur_filmes (que_cursor IN NUMBER);
    ...
END pkg_filmes;
```

No corpo da *package*, a lógica das *procedures* é colocada (apenas as partes alteradas são mostradas).

```
CREATE OR REPLACE PACKAGE BODY pkg_filmes AS
   -- Implementação da lógica de funções e procedures
   FUNCTION consulta_filme (filme_id filmes.filme_id%TYPE)
   RETURN TIPO_REG_FILMES
    ...
   -- Abre os cursores de pesquisa por diretor ou país. Se
fecha_e_abre = T ou = NULL e o
   -- cursor estiver aberto, fecha e reabre.
   PROCEDURE open_cur_filmes (que_cursor IN NUMBER,
fecha_e_reabre IN BOOLEAN,
      diretor_ou_pais IN VARCHAR2)
   IS
   BEGIN
      CASE que_cursor
         WHEN cur_diretor THEN
            IF (fecha_e_reabre OR (fecha_e_reabre IS NULL)) AND
cur_filmes_diretor%ISOPEN THEN
               CLOSE cur_filmes_diretor;
               OPEN cur_filmes_diretor(diretor_ou_pais);
            ELSIF NOT cur_filmes_diretor%ISOPEN THEN
               OPEN cur_filmes_diretor(diretor_ou_pais);
            END IF;
         WHEN cur_pais THEN
            IF (fecha_e_reabre OR (fecha_e_reabre IS NULL)) AND
cur_filmes_pais%ISOPEN THEN
               CLOSE cur_filmes_pais;
               OPEN cur_filmes_pais(diretor_ou_pais);
            ELSIF NOT cur_filmes_pais%ISOPEN THEN
               OPEN cur_filmes_pais(diretor_ou_pais);
            END IF;
         ELSE
            NULL;
      END CASE;
   END;

   -- Abre os cursores de pesquisa por diretor ou país. Se
fecha_e_abre = T e o cursor estiver aberto,
   -- fecha e reabre. Se ano_ini ou ano_fim = NULL, mas não ambos,
faz o parâmetro = NULL receber o
   -- valor do outro parâmetro.
   PROCEDURE open_cur_filmes (que_cursor IN NUMBER,
```

A seguir, os exemplos de uso da *package pkg_filmes* com os novos cursores e *procedures*.

```
CREATE OR REPLACE PROCEDURE print_filmes (
   reg pkg_filmes.tipo_reg_filmes
)
IS
BEGIN
   DBMS_OUTPUT.PUT_LINE('Título:  ' || reg.titulo);
   DBMS_OUTPUT.PUT_LINE('Diretor: ' || reg.diretor);
   DBMS_OUTPUT.PUT_LINE('País:    ' || reg.pais);
   DBMS_OUTPUT.PUT_LINE('Ano:     ' || reg.ano);
   DBMS_OUTPUT.PUT_LINE('Duração: ' || reg.duracao);
   DBMS_OUTPUT.NEW_LINE;
END;

DECLARE
   reg pkg_filmes.cur_filmes_diretor%ROWTYPE;
BEGIN
   pkg_filmes.open_cur_filmes(pkg_filmes.cur_ano, TRUE, 1960,
1990);
   LOOP
      FETCH pkg_filmes.cur_filmes_ano INTO reg;
      EXIT WHEN pkg_filmes.cur_filmes_ano%NOTFOUND;
      print_filmes(reg);
   END LOOP;
   DBMS_OUTPUT.NEW_LINE;
   DBMS_OUTPUT.NEW_LINE;
   -- O cursor pkg_filmes.cur_filmes_ano não foi fechado
   pkg_filmes.open_cur_filmes(pkg_filmes.cur_ano, TRUE, 1981,
NULL);
   LOOP
      FETCH pkg_filmes.cur_filmes_ano INTO reg;
      EXIT WHEN pkg_filmes.cur_filmes_ano%NOTFOUND;
      print_filmes(reg);
   END LOOP;
   pkg_filmes.close_cur_filmes(pkg_filmes.cur_ano);
   -- O cursor já foi fechado (veja o comando anterior). Vou fechá-lo
de novo.
   pkg_filmes.close_cur_filmes(pkg_filmes.cur_ano);
   DBMS_OUTPUT.NEW_LINE;
   DBMS_OUTPUT.NEW_LINE;
   pkg_filmes.open_cur_filmes(pkg_filmes.cur_pais, TRUE, 'Brasil');
   LOOP
```

O resultado da execução do bloco anônimo acima está no material de apoio.



Atividade Extra

Em geral, não é permitido que um *trigger* DML consulte dados da tabela a qual está vinculado por causa de um fenômeno chamado *dirty reads* (leituras sujas). No entanto, há uma forma de fazê-lo. Pesquise o que é e como podem ser evitadas leituras sujas no SGBDR Oracle.

Referência Bibliográfica

- FEUERSTEIN, S. **Oracle PL/SQL Programming**. 6ª Ed., O'Reilly, 2014.
- PUGA, S., FRANÇA, E. e GOYA, M. **Banco de Dados: Implementação em SQL, PL SQL e Oracle 11g**. São Paulo: Pearson, 2014.
- Gonçalves, E. **PL/SQL: Domine a linguagem do banco de dados Oracle**. Versão Digital. Casa do Código, 2015.
- GROFF, J. R., WEINBERG, P. N. e OPPEL, A. J. **SQL: The Complete Reference.** 3ª Ed., Nova York: McGraw-Hill, 2009.
- ELMASRI, R. e NAVATHE, S. B. **Sistemas de Banco de Dados**. 7ª Ed., São Paulo: Pearson, 2011.

[1] Observe que a consulta consiste em uma junção total de três tabelas. Como o SGBDR sabe que linhas devem ser trancadas? Quando uma junção é realizada, o SGBDR registra que linha de que tabela é utilizada para cada linha do resultado da consulta. Com isto, ele consegue saber que linhas deve trancar.

Ir para exercício